COMPUTER SCIENCES AND SOFTWARE ENGINEERING
AUBURN UNVERSITY

COMP 2710
Software Construction

Summer 2013

Lab 2
A Simple Distributed Messaging System

Due: July 10, 2013

Points Possible: 100
Due: Written Portion: July 3rd, 2013 by 11:55 pm turned in via CANVAS SUBMISSION (25 points)
Program: July 10th, 2013 by 11:55 pm turned in via CANVAS SUBMISSION (75 points)

No late assignments will be accepted.
No collaboration between students. Students should NOT share any project code with each other. Collaborations in any form will be treated as a serious violation of the University's academic integrity code.

Goals:

- To perform Object-Oriented Analysis, Design, and Testing
- To develop a non-trivial application using classes and constructors
- To learn distributed communication methods and develop a simple but useful distributed messaging system
- To learn to use file I/O and take advantage of the Network File System (NFS)
- To learn the use of arrays and maintain the number of elements used

Process – 25 points:
Create a text, doc, or .pdf file named "<username>-2p" (for example, mine might read "lim-2p.txt") and provide each of the following. Please submit a text file, a .doc file or .pdf file (if you want to use diagrams or pictures, use .doc or .pdf). You are free to use tools like Visio to help you, but the final output needs to be .txt, .doc, or .pdf.

1. **Analysis:** Prepare use cases. Remember, these use cases describe how the user interacts with a messaging system (what they do, what the systems does in response, etc.). Your use cases should have enough basic details such that someone unfamiliar with the system can have an understanding of what is happening in the messaging system interface. They should not include internal technical details that the user is not (and should not) be aware of.
2. **Design**:
   a. Create a Class Diagram (as in Lab 1). Be sure to include:

1) The name and purpose of the classes
2) The member variables and the functions of the class
3) Show the interactions between classes (for example, ownership or dependency)
4) Any relevant notes that don't fit into the previous categories can be added

b. Create the data flow diagrams. Show all the entities and processes that comprise the overall system and show how information flows from and to each process.

3. **Testing**: Develop lists of *specific* test cases OR a driver will substitute for this phase:

1) For the system at large. In other words, describe inputs for "nominal" usage. You may need several scenarios. In addition, suggest scenarios for abnormal usage and show what your program should do (for example, entering a negative number for a menu might ask the user to try again).

2) For each object. (Later, these tests can be automated easily using a simple driver function in the object)

4. Implement the distributed messaging system

5. **Test Results**: ***After developing the distributed messaging system***, actually try all of your test cases (both system and unit testing). Actually show the results of your testing (a copy and paste from your program output is fine – don't stress too much about formatting as long as the results are clear). You should have test results for every test case you described. If your system doesn't behave as intended, you should note this. Note: Driver output will substitute for this phase.

Program Portion:

The proliferation of Internet applications in smart phones, laptops, ipads, and desktops shows the importance of communications among computing devices. One of the key functionalities is the ability to send and receive messages from one computer to another over the network. It is interesting to learn how a simple messaging system can provide this functionality. In this Lab 2, you will analyze, design, implement and test a simple distributed messaging system. It will have a simple text-based user interface that allows multiple users to send and receive messages to each other. In this lab project all the users must be implemented in distributed address space, i.e. each user must execute on a different computer and message can be sent from one user to another through a common file. The common file is shared through the Network File Server (NFS).

What is a distributed messaging system?

You probably have used a distributed messaging system over the Internet. Although there are many different types of messaging systems, you will implement the message system that allows users to send and receive messages to and from another user executing on a different computer, by naming the receiving user. For the purpose of this lab project, a distributed messaging system is a program (or collection of programs) that does the following:

1) A user *Bill* can send and receive messages to and from user *Jane* by naming his friend *Jane*. Although the user need not know this, the messages are sent and received through a *message channel* identified by the message channel name *Bill-Jane*. The message channel is an implementation that is hidden from the user. The message channel must be bi-directional, i.e. messages can flow from *Bill* to *Jane* and vice versa.
2) *Jane* can also receive and send messages to *Bill* by simply naming *Bill* as her *friend*. Although hidden from the user, the messaging system will use the same message channel *Bill-Jane* as in the previous bullet. In order to ensure the message channel has a unique name, put the names in lexicographical order before combining them with a "-".
3) A user may send and receive messages to and from many other users. There is no limit to the number of users, but for all practical purposes in the lab project, you can set the limits to 10. You need to use array to keep track of the friends with which the user is communicating and keep track of the number of elements in the array that is actually used. However, at any time, a user can send and receive messages from only one friend.
4) For each friend that a user is communicating with, there must be a message channel. So if a user is communicating with five friends, he will use five different message channels.
5) Since a message channel is bi-directional, it is implemented by using two *message pipes*, one for each direction. For example, the message channel *Bill-Jane* is implemented as two messages pipes, *Bill_Jane* and *Jane_Bill*. The message pipes *Bill_Jane* is for forwarding messages sent by Bill and received by Jane and so on.
6) Message pipes are implemented using files. So if Bill sends a message to Jane, the messaging system will write the message into the file *Bill_Jane* representing that pipe. Since the file is maintained by NFS, the user Jane will also see the same file *Bill_Jane*, although it is executing on a different computer. For Jane to receive the message, the messaging system will read the message from the file *Bill_Jane*.
7) When a user receives messages, *all* messages will be displayed, the user will receive all previous messages that another user sent earlier. The message itself will not contain the sending user name. This is because at any time, a user is communicating with only one friend.
8) Once a message is received (read) by the receiving user, the message will be erased from the message pipe.
9) At most two users can share each message channel.
10) A user may send multiple messages to a friend before the friend receives the message.

Message format

For this assignment, design your own chat messages using any appropriate format. The main requirements are that the message must *not* contain the user's name, only the text of the message. However, each message must be of variable length. Each message can be separated easily from another message using "\n" to separate the messages. This allows multiple messages to be sent by a user before the receiving user receives the message.

The user interface

Write a menu-based and text-based user interface for interacting with the distributed messaging system. When the message system is first started, it will prompt for the user's name. After the user's name is entered, the system will print a banner welcoming the user.

```
Please enter user name: Bill

        ============================================================
        |         Welcome to the Messaging System, Bill!           |
        ============================================================
```

The main menu has (at least) 4 options to choose from as shown below. *Abbreviate the menu so that all the 4 options can be shown in only one or two lines.* The user interface will accept the option and all related inputs from the user, perform the operation and then repeatedly shows this menu of options again and prompts for the next option.

- **Select friend (f)**: When the user enter option 'f', the program will then prompt for the name of the friend and then stores that name as the current receiver. This option can also be used to switch the friend's name.
- **Send a message (s)**: When the user enters the option 's', the program will prompt for the message and send message to the *current message channel* to the current friend.
- **Receive messages (r)**: When the user enters the option 'r', the program will display all messages that are in the *current message channel* from the current friend. If there is no message, it will display "No new message".
- **Quit this messaging system (q)**: When the user enters the option 'q', the program will exit normally.

The user interface must check for correct input value from the users. If there is any error, e.g. sending a message *without* selecting a friend, then the program must display the appropriate error message and continue to prompt for the correct input. Your program must not exit or terminate when there is an incorrect input value.

The name of your program must be called <username>_2.cpp (for example, mine would read "lim_2.cpp"

Use comments to provide a heading at the top of your code containing your name, Auburn Userid, and filename. You will lose points if you do not use the specific program file name, or do not have a comment block on **EVERY** program you hand in.

Your program's output need not exactly match the style of the sample output (see the end of this file for one example of sample output).

Important Notes:
You must use an object-oriented programming strategy in order to design and implement this distributed messaging system (in other words, you will need write class definitions and use those classes, you can't just throw everything in main()). A well-done implementation will produce a number of robust classes, many of which may be useful for future programs in this course and beyond. Some of the classes in your previous lab project may also be re-used here, possibly with some modifications. Remember good design practices discussed in class:

a) A class should do one thing, and do it well
b) Classes should NOT be highly coupled
c) Classes should be highly cohesive

- You should follow standard commenting guidelines.

- You DO NOT need any graphical user interface for this simple, text-based application. If you want to implement a visualization of some sort, then that is extra credit.

Error-Checking:

You should provide enough error-checking that a moderately informed user will not crash your program. This should be discovered through your unit-testing. Your prompts should still inform the user of what is expected of them, even if you have error-checking in place.

Submit your program through the Canvas system online. If for some disastrous reason Canvas goes down, instead e-mail your submission to TA – Sankari Anupindi – at sza0033@tigermail.auburn.edu. Canvas going down is not an excuse for turning in your work late.

You should submit the two files in digital format. No hardcopy is required for the final submission:

<username>_2.cpp
<username>_2p.txt (script of sample normal execution and a script of the results of testing)

Sample Usage:

The following is executed in tux187:

> lim_2

```
        ============================================================
        |           Auburn Distributed Messaging System!           |
        ============================================================

        Please enter user name: Bill

        ============================================================
        | Welcome to Auburn Distributed Messaging System, Bill!    |
        ============================================================

Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: f

Please enter friend's name: Jane

        ============================================================
        |                     Connected to Jane                    |
        ============================================================

Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: s

Enter message: Hello Jane!

        ============================================================
        |                  Message sent to Jane                    |
        ============================================================

Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: f

Please enter friend's name: Tom

        ============================================================
        |                     Connected to Tom                      |
        ============================================================

Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: s

Enter message: Good Morning Tom!

        ============================================================
        |                   Message sent to Tom                     |
        ============================================================

Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: s

Enter message: See you in class.

        ============================================================
        |                   Message sent to Tom                     |
        ============================================================
```

```
Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: r

Message(s) from Tom:
Great! Talk to you after class.

Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: f

Please enter friend's name: Jane

        ============================================================
        |                    Connected to Jane                     |
        ============================================================

Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: r

Message(s) from Jane:
Hey Bill, good to hear from you!

Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: q

        ============================================================
        |        Thank you for using the messaging system          |
        ============================================================
```

The following is executed in tux192:

> lim_2

```
        ============================================================
        |           Auburn Distributed Messaging System!           |
        ============================================================

        Please enter user name: Tom

        ============================================================
        | Welcome to Auburn Distributed Messaging System, Tom!   |
        ============================================================

Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: f

Please enter friend's name: Jane

        ============================================================
        |                     Connected to Jane                    |
        ============================================================

Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: s

Enter message: Hi Jane, when are you heading to campus?

        ============================================================
        |                  Message sent to Jane                    |
        ============================================================

Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: f
```

```
Please enter friend's name: Bill

        ============================================================
        |                    Connected to Bill                     |
        ============================================================

Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: r

Message from Bill:
Good Morning Tom!
See you in class.

Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: s

Enter message: Great! Talk to you after class.

        ============================================================
        |                  Message sent to Bill                    |
        ============================================================


Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: f

Please enter friend's name: Jane

        ============================================================
        |                    Connected to Jane                     |
        ============================================================

Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: r

Message(s) from Jane:
About quarter till

Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: q

        ============================================================
        |        Thank you for using the messaging system          |
        ============================================================
```

The following is executed in tux189:

> lim_2

```
        ============================================================
        |           Auburn Distributed Messaging System!           |
        ============================================================

        Please enter user name: Jane

        ============================================================
        | Welcome to Auburn Distributed Messaging System, Jane!    |
        ============================================================
```

Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: **f**

Please enter friend's name: **Tom**

```
        ============================================================
        |                    Connected to Tom                      |
        ============================================================
```

Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: **r**

Message from Tom:
Hi Jane, when are you heading to campus?

Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: **s**

Enter message: **About quarter till**

```
        ============================================================
        |                  Message sent to Tom                     |
        ============================================================
```

Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: **f**

Please enter friend's name: **Bill**

```
        ============================================================
        |                   Connected to Bill                      |
        ============================================================
```

Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: **r**

Message from Bill:
Hello Jane!

Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: **s**

Enter message: **Hey Bill, good to hear from you!**

```
        ============================================================
        |                  Message sent to Bill                    |
        ============================================================
```

Select Friend (f), Send (s), Receive (r), Quit (q), Enter option: **q**

```
        ============================================================
        |        Thank you for using the messaging system          |
        ============================================================
```